Kompira：シンプルで軽量な IT 運用自動化プラットフォーム

服部健太† 溝江真也† 三角正樹†

仮想化環境やクラウドの浸透によって，IT システムの利用は高度化しており，それにともないシステム運用の負荷も高くなっている．このような状況の中，多くの現場では，突然のトラブル対応や頻繁な設定変更の依頼などに追われて，エンジニアが疲弊してしまうといった問題がある．現場では，監視ツールやチケット管理ツールなど，運用のための様々なツールを導入したり，個別のスクリプトやマクロを駆使したりするなどして，効率化の努力を行っているところも多いが，最終的に，これらツール間のやり取りを行うのは人手であるため，なかなか負荷軽減につながらない．我々はこのような IT システムの運用現場を少しでも効率化することを目指して，IT 運用自動化プラットフォーム Kompira を開発している．従来から数多くある統合運用管理ツールと異なり，Kompira では運用自動化に必要な最小限の機能に的を絞ることで，シンプルで軽量なプラットフォームを実現している．運用業務とは数多くのツール間を連携させることであるととらえ，Kompira はそれらをジョブフローと呼ぶ運用手順を記述したスクリプトで結びつける．本稿では，Kompira の特長や設計と実装について説明し，実際の適用事例を紹介する．

Kompira: Simple and Lightweight Automation Platform for IT System Operations

KENTA HATTORI† SHINYA MIZOE† MASAKI MISUMI†

Daily activities for IT System operations are getting harder because of heavier use of IT system with virtualization and cloud computing technologies. There are problems with impoverished engineers in the day to day activities, because they are overloaded with a lot of tedious works such as sudden trouble shooting, frequent configuration changes and so on. Although they make efforts to improve their operations by adopting various management tools or devising scripts and macros for automation, their effect is limited because operations among those tools are executed manually in the end. We are developing an automation platform called Kompira. The goal of Kompira is to optimize IT system operations and management. Unlike many conventional tools for integrated operational management, Kompira focuses minimal function set required for automatic operations. We consider operational activities as filling gaps between various tools. Kompira combines those tools by scripts called jobflow that express operational procedures. This paper describes features, design, and implementation of Kompira, and then we introduce several application examples of actual cases.

1. はじめに

IT システムの運用自動化は古くて新しいテーマである．これまでに様々なツールが開発されてきており，昨今では仮想化環境やクラウド環境の浸透による運用負荷の増大を背景として，自動化は益々注目を集めてきている．システム運用の業務は，障害の監視と一次対応，デプロイや構成管理，設定変更，インシデントの記録，問題発生時のエスカレーション，証明書の更新，バックアップ，パッチ適用やバージョンアップ対応などなど，非常に多岐にわたっている．運用現場では，これらの業務を効率化するために個別のツールを導入していることも多いが，実際の運用手順では様々なツール間をまたがっての作業が必要で，最終的には，あるツールの出力結果を別のツールの入力に貼り付けるといった泥臭い作業を運用担当者が人手で行うといったことになりがちである．

大規模なサービスを提供している事業者など，一部の先進的な運用現場では，OSS ツールを組み合わせて独自のシステムを構築したり，スクリプトを作成したりするなどして，運用の効率化や自動化を図っているところもあるだろう．しかしながら，多くの中小規模のシステムの運用現場では，技術力や資金力に乏しいため，自前で運用自動化のシステムを構築したり，高価な大手ベンダー製の運用自動化ツールを導入したりすることができず，結局は人手で運用していることが多い．あるいは，せいぜい現場の担当者が独自に作成したスクリプトによる部分的な自動化に留まっていることが多い．

Kompira[1]は，運用監視サービス事業者としての我々の経験にもとづいて開発を行っている運用自動化のための基盤ツールである．運用監視サービスの現場では，監視対象のサーバに障害が発生すると，手順書にしたがってオペレータが障害対応を行っていくという定型的な作業が多くの割合を占めている．運用担当者が行っている日々の定型的な業務の多くは，独自の DSL である**ジョブフロー言語**を用いて記述することで自動化することが可能である．

Kompira のジョブフロー実行エンジンは，コンパイルされた中間コードを逐次実行し，対象のシステムにリモートにログインし，決められた手順のコマンドを実行し，結果によって，別のコマンドを実行するといったことを行う．ジョブフロー言語は並行プロセスをベースにしており，チャネルを用いて外部ツールとの連携処理を簡潔に記述することができる．また，Kompira は運用に必要な種々の情報

† フォースクーナ株式会社
ForSchooner Inc.

を定義し格納するためのオブジェクトストレージを備えており，ジョブフローからこれらのデータに統一的にアクセスし操作することが可能である．

このように Kompira は，様々なツールや管理対象となる多数のサーバ群にまたがる運用処理をジョブフローによって結びつけ，システム運用全体を自動化するためのプラットフォームである．運用現場のオペレータが，自分たちの日々の定型的な作業をジョブフローによって自動化していくことで，繰り返しの定型作業が無くなり，現場の負荷軽減や運用コストの削減につながることが期待できる．

本稿では，まず，Kompira 開発の背景として IT システムの運用現場が抱えている問題を説明し（2 章），これまでに用いられてきた運用業務のためのツールをいくつか紹介する（3 章）．次に，我々の開発している Kompira の特長と設計思想について述べてから（4 章），システムの概要（5 章）と，代表的な機能の内容と実装について触れる（6 章）．そして，Kompira の実際の適用事例を紹介してから（7 章），Kompira 適用の効果や現状の課題などを検討し（8 章），最後にまとめをおこなう（9 章）．

2. IT システム運用現場の抱える問題

システムの運用とは，サービスを安定稼働させ，サービスを常に提供可能な状態に保つことが目的であり，その重要性はシステム開発に勝るとも劣らない．しかしながら，IT システムの運用現場の実態は，なかなか表だって語られることの少ないテーマであり，運用現場の抱えている問題が広く一般に理解されているとは言い難い状況である．本章では，我々がこれまで運用監視サービスを提供してきた経験にもとづいて，多くの運用現場が抱えている問題を指摘する．

2.1 疲弊していく現場

波多野[2]は，運用現場の抱える問題として，(1) 高負荷，(2) 属人的，(3) 費用対効果の見えない，という 3 点を指摘している．実はこれら 3 つの問題点はその要因が密接に絡み合っている．

最も大きな要因の 1 つは，システムの開発段階で想定していなかった顧客からの要求や，システムのトラブルなどを「運用でカバー」するといった形で現場に押し付けられることである．その結果，業務範囲が明確に規定されず，多岐にわたってしまうため，現場が高負荷となってしまう．さらに，業務タスクやフローが数多くあるにもかかわらず，高負荷のため，これらがきちんとドキュメント化されることは滅多に無く，結果として知識が属人化してしまう．少し気の利いた担当者だと，自分の業務を自動化するスクリプトを作成することで業務を効率化する場合もあるが，そのスクリプトは作った本人専用のものとなってしまうことも多い．そのような，いわゆるオレオレスクリプトが 2012 年 6 月に発生したファーストサーバの事故[3]の原因の 1 つとなったことは記憶に新しい．

さらに，経営者側からは，運用業務は利益を生み出さないコストセンターとみなされてしまうため，現場は常にコスト削減，業務効率化の圧力にさらされる．きちんと対応してあたりまえ，ミスをすると責められる．責められないまでも，同じミスを繰り返さないような対策を求められる．そしてその対策はたいていの場合，二重チェックをする，チェックシートを作るなどといった業務負荷を増やす方向に作用するため，ますます現場は忙しくなってしまう，というような悪循環に陥るのである．

2.2 優秀な人材の枯渇

運用現場の抱えるもう 1 つの大きな問題として，優秀な技術者がなかなか定着しないという点が挙げられる．新規のシステム開発のように華やかなイメージを持つ業務に比べると，運用保守は繰り返しの作業が多く，地味で退屈な業務であると思われがちである．さらに前節で述べたように，高負荷でなかなか休みが取れない，また，特に 24 時間 365 日の稼働が期待されているシステムの運用では，ひとたび障害が起こると緊急の対応を要請されるため，深夜・休日でもなかなか気が休まる暇がない．要するに労働環境が恵まれているとは言い難い．このような背景から，創造力や技術力があって，意欲的な人材ほど，運用保守の現場から離れていってしまうのである．実際には，トラブル対応などは幅広い知識と経験が解決につながることが多く，そのような意味でも，優秀な人材が離れてしまう問題は大きいといえる．

運用業務のミッションには，障害時のトラブルシュートや，バックアップ，設定管理等のサービス維持業務以外に，パフォーマンス向上や安定化のための予防的保守など，サービス品質向上のための改善業務も含まれる．本来ならば，優秀な人材をこそ改善業務に専念させ，サービス品質の向上，ひいてはビジネスの競争力を高めることに貢献してもらうことが重要である．しかし，実際には日々の維持業務で手一杯となってしまい，さらにシステムの改善を提案できるような優秀な技術者が不足しているため，なかなか改善業務にまで手をつけられないという現場は多い．

2.3 開発と運用のかい離

開発段階で想定していなかった要求やシステムのトラブルなどは，運用チームだけで根本的な解決を行うことが難しく，開発側に何らかの対応を依頼する必要があることも多い．最近では，開発（Development）と運用（Operation）を組み合わせた **DevOps**[4]というキーワードが注目を集めており，開発と運用が協力しあうことで，迅速にシステムの改善につなげていこうという機運が高まっている．DevOps 運動は興味深い概念ではあるが，多くの運用現場では，そもそも別のベンダーが開発したシステムや，開発者が既に居なくなってしまったシステムの運用を任されたりしているのが実情で，そもそも DevOps が成立しえない．

このような状況では，アプリケーションのバグなどで，頻繁にクラッシュしたりディスク領域にゴミが溜まって定期的なクリーンアップが必要なシステムでも，改修を開発元になかなか対応してもらうことができず，結局はアプリケーションをリブートしたり，不要なファイルを削除したりといった開発側の後始末を運用現場が担わされる．こうして現場の負荷がますます高くなるのである．

2.4 まとめ

結局，IT システムの運用現場が抱える問題をまとめると，次のようになる．IT システムの高度化，複雑化によって，システムの運用・保守にかかるコストや現場の負荷は増大している．さらに，システム稼働当初からは想定していなかったようなトラブルや要求が「運用でカバー」することを押しつけられ，現場の負荷がさらに高まる．一方で経営側からは，コスト削減や業務効率化の圧力にさらされるが，業務内容を正当に評価される機会は少なく，現場の士気は低下していく．本来なら，システムの改善業務に多くの時間を割きたいところだが，優秀な人材の不足や，高負荷で時間が足りない状況では，目の前の作業をこなすことに精一杯となり，現場が疲弊していく．以上が現場の抱える問題なのである．

3. 従来の運用自動化ツール

冒頭にも述べたとおり，運用自動化のツールは数多くある．これらを大別すると，統合的な運用管理ツールと，デプロイやテスト，ビルド作業等，特定の作業の自動化に特化したツールとに分けられる．以下では，それぞれのツールについて，代表的なものを取り上げて紹介する．

3.1 統合運用管理ツール

統合運用管理ツールは，機器の監視やジョブ管理，リソース管理など，IT システムの運用管理に必要なツールを統合して，一元的に管理するためのツールである．大手ベンダーが提供するものから，オープンソースのものまで数多く存在する．統合運用管理ツールでは，運用業務の自動化はジョブ管理機能が担う．スナップショットの取得や，データのバックアップ，データ集計など，自動化する業務を構成する一つひとつの処理を「ジョブ」として定義していく．さらに，これらジョブの実行順序や依存関係を「ジョブネット」といった形で定義することで業務を自動化する．ジョブネットはスケジューラによって，指定した曜日や日時によって起動したり，メール受信などのイベントを契機に起動したりする．ベンダーの製品では，ジョブネットは GUI による作成画面から直感的な操作で作成できるような機能を提供していることが多い．

この分野では，日立の JP1[5]がデファクトスタンダードの地位を占めており，国内では広く使われているツールである．日本の商慣行にきめ細やかに対応しており，いわゆる五・十日の設定や，実行予定が休業日にあたる場合，前後に振り替えるなど，スケジューリング機能が充実していることが特長の 1 つである．JP1 以外では，富士通の SystemWalker, NEC の WebSAM, IBM の Tivoli などがある．オープンソースでは，Hinemos[6]が有名であり，他にも Crane[7]といったものがある．

大手ベンダーが提供している統合運用管理ツールは，文字通り様々なツールが統合された製品であり，ジョブ管理だけでなく，システム監視，リソース管理，セキュリティ管理など，様々なツールから構成されている．

ところで，ジョブ管理機能はもともと，企業が抱える数多くの業務バッチを管理し，自動化するためのツールとして発展してきた経緯がある．このため，第 1 章で述べたような IT システムの運用作業をジョブ管理機能で実現するには無理がある．一方，最近では，こうした IT システムの運用作業を自動化するためのツールとして，**ランブックオートメーション**（RBA）が注目を集めている[8]．ヒューレット・パッカードの HP Operations Orchestration や BMC ソフトウェアの BMC Atrium Orchestrator, NetIQ の NetIQAegis などがある．

3.2 特定分野における自動化ツール

Puppet[9]はサーバの構築やシステムの設定管理を自動化するためのツールである．専用の外部 DSL を用いて管理対象となるサーバのあるべき設定を記述することで，対象のシステムを常に望ましい設定状態に保つことができるようになる．たとえば，誰かが一時的な対応によって設定ファイルを変更した場合でも，それを検出して元通りの状態に戻すことができる．似たようなツールとしては Chef[10]があるが，こちらは Ruby の内部 DSL を用いて，設定ファイルを記述するため，より柔軟な処理が可能である．

デプロイのための自動化ツールとしては，Capistrano[11]や Fabric[12]などが知られていて，それぞれ Ruby と Python で実装されている．アプリケーションのデプロイ手順を，DSL で記述することで，コマンドラインから簡単にデプロイ作業を実行することができる．また，対象のサーバには SSH で自動ログインして処理を行うため，特別なエージェントをあらかじめインストールしておく必要がない．

RightScale[13]はクラウドの運用管理を自動化するためのサービスである．サーバの構成管理だけでなく，負荷状況に応じたプロビジョニング，スケールアウト／インといった作業を自動化できる．自動化の処理は，bash や Perl, Ruby や Python などのスクリプト言語を用いて記述する．

その他，運用管理というよりは，むしろ開発に向いたものであるが，継続的インテグレーションのためのツールとして Jenkins[14]が有名である．これは，ソースコードをレポジトリから取得してビルドとテストを行うといった，開発に必要な一連の作業を自動化することができる．また，ビルド結果のレポートやテスト結果のカバレッジレポートなどを出力することもできる．

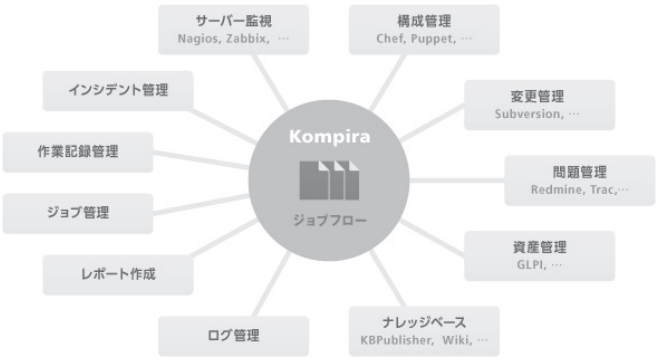

図 1　各種運用ツールのハブとしての Kompira

4. Kompira の特長と設計思想

Kompira は，3 章で紹介した統合運用管理ツールや，特定分野に特化した自動化ツールとは異なる考え方にもとづく運用自動化ツールである．本章では，我々が Kompira を開発する上で重要視している設計思想や Kompira の特長について説明する．

4.1 各種ツールのつなぎとしての役割

先述のとおりシステムの運用業務は，非常に多岐にわたっている．運用現場では，これらの作業負荷を少しでも軽減するために，様々なツールを利用している．たとえば，オープンソースのツールに限れば，サーバの障害監視なら，Nagios や Zabbix，問題発生時のチケット管理ならば Redmine や Trac といったツールなどである．大手ベンダーの統合運用管理ツールは，障害監視やチケット管理など，運用業務に関わる機能を丸ごと取り込んで巨大なパッケージとして提供しているが，Kompira では，様々なツールを連携させるための「ハブ」あるいは「つなぎ」としての役割に特化している（図 1）．このため，Kompira 自身は軽量でシンプルなツールなものとなる．すなわち，障害監視やチケット管理などの部分は，既存の優れたツールに任せてしまい，Kompira はそれらを連携させる部分に集中することで，Kompira 自体の開発規模を膨らませることなく，全体として統合的な運用管理を実現することができるのである．また，様々なツールとの連携を重視しているため，既存の運用システムを大きく入れ替えることなく，Kompira だけ導入するといったことも可能となる．これによって Kompira 導入の敷居を下げることができる．

4.2 できるところから自動化

我々は，すべての業務を Kompira で自動化するべきとは考えてはいない．運用の自動化にあたっては，人間が行うべき作業と，自動化すべき作業とを分けて考えることが重要である．たとえば，高度な判断が求められるような業務や重要なファイルの削除等は，自動化することが難しかったり，心理的に抵抗があったりするかもしれない．そのようなケースを考慮して，Kompira では自動化の途中で，人間による承認や確認，判断などの処理を差し挟むことが可能である．

また，障害時の復旧作業などを自動化することを考えたとき，想定外の障害や状況のもとで無理に自動処理を継続しようとすると，却って事態を悪化させてしまうことになる．そこで，Kompira では，フェイルセーフの原則にのっとって，実行中に予期せぬエラーが発生した場合は，ジョブフローを異常終了するようにしている．

4.3 運用業務の進化をサポート

運用の細かいノウハウは運用現場に蓄積されているため，業務を効率化するには現場のエンジニアが主導して行うのがよい．Kompira のジョブフロー言語は，bash や perl などのスクリプト言語に触れたことのあるエンジニアならば短期間で習得できるよう，シンプルな設計にしている．エンジニアにとって Kompira がブラックボックスとなるのではなく，自分たちの抱えている業務を自ら自動化していけるようなツールを目指している．たとえば，自動化されていない想定外の対応作業が発生した場合でも，それをきっかけにしてエンジニアがその場で，自動化するジョブフローを作成すれば，以後，同じ作業が発生した場合には，同じことを人手で繰り返す必要がなくなる．このように，現場が Kompira を用いて少しずつ運用自動化を育てていくことで，業務負荷が下がり，その時間を用いてさらなる効率化のために，自動化のジョブフローを作成する，といった好循環が発生することが期待できる．

5. Kompira のシステム概要

図 2 に Kompira のシステム概要を示す．Kompira システムは，ユーザーへの WebUI の提供とジョブフローの実行管理を行う Kompira サーバ，そして，Kompira サーバからコマンド実行の指示を受け，管理対象のサーバに対して SSH 経由でリモートコマンドを実行する Kompira ジョブマネージャの 2 つから構成される．

5.1 Kompira サーバ

Kompira サーバは，Web アプリケーションであり，Django フレームワークを用いて実装している．Web サーバ（Apache），データベース（MySQL）に加えて，ジョブマネージャに対するコマンド実行メッセージをやり取りするためのメッセージキュー（RabbitMQ），および，ジョブフローの実行プロセスを管理するための Kompira エンジンがデーモンプロセスとして動作している．

5.2 Kompira ジョブマネージャ

ジョブマネージャは起動すると，Kompira サーバで動作しているメッセージキューの待受けポートに接続を行い，コマンド実行の指示を待つ．したがって，管理対象のサーバ群がファイアウォールで保護されていて，Kompira サーバから各管理対象サーバに直接 SSH ログインすることが禁止されている環境でも，メッセージキューのポートに対して外向きのアクセスだけ許可しておけば，Kompira を運

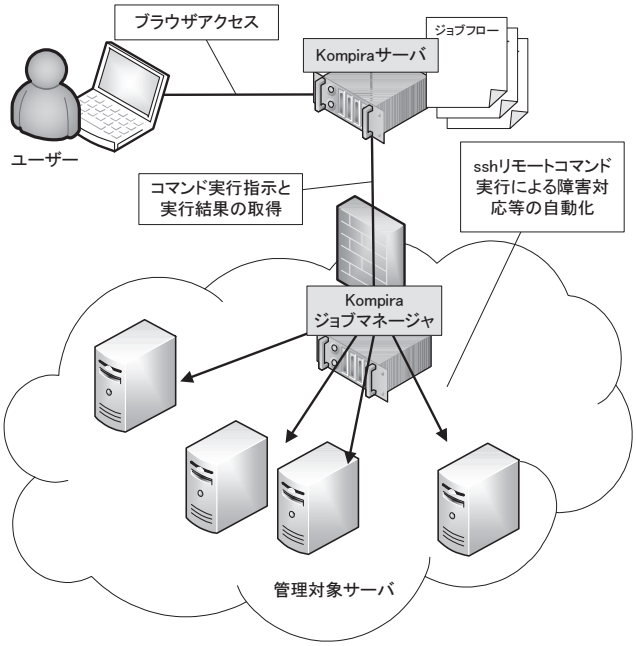

図 2　Kompira のシステム概要

用することができる．管理対象サーバの運用を行うオペレータは，社内ネットワークに直接アクセスできない環境にいる場合でも，Kompira を利用することで対象サーバの操作を行うことが可能である．

5.3 システム構成のバリエーション

図 2 では，ジョブマネージャは Kompira サーバとは別のネットワークセグメント上で動作している構成を示した．Kompira サーバをイントラネットなどファイアウォールの内側に設置する場合には，バリエーションとしてシンプルに 1 台の機器に Kompira サーバとジョブマネージャを同居させることもできる．一方で，複数のネットワークセグメントを管理する場合には，1 つの Kompira サーバに対して，各ネットワークセグメントにジョブマネージャを置く，といった構成も可能である．この場合，Kompira サーバとジョブマネージャの関係は 1 対多となる．

6. 諸機能と実装

本章では，Kompira が提供する機能のうち，代表的なものを紹介し，必要に応じてその実装について説明する．

6.1 オブジェクトストレージ

Kompira では，ジョブフローや管理対象機器の情報などの各種データは，すべてオブジェクトとして Kompira サーバ上の Kompira オブジェクトストレージに格納される．

6.1.1 オブジェクトの型

オブジェクトは，属性値を保持するためのフィールドを備えており，各オブジェクトがどのようなフィールドを含んでいるかは，そのオブジェクトの型によって規定される．特徴的なのは，型を定義するデータ自体も，Kompira のオブジェクトであり，型オブジェクト（TypeObject）としてオブジェクトストレージに格納されている点である．ちなみに，型オブジェクトの型もまた，型オブジェクトである．これはオブジェクト指向プログラミングにおけるクラスとメタクラスの関係に相当する．

ユーザーは，型オブジェクトを定義することで，自由に新しい種類のデータを定義することができる．ユーザーが新しい型を定義するには，オブジェクトが備えるべきフィールドの名称と表示名，さらに，そのフィールドが持つデータの種類（String や Integer, Boolean, Text, EmailAddress など）を WebUI の新規作成画面から指定する．たとえば，フィールドとして，住所，氏名，電話番号，E メールアドレスを持つ「連絡先」という型オブジェクトは，以下のように定義できるだろう．

フィールド名	フィールド表示名	フィールド種別
name	氏名	String
address	住所	Text
phone	電話番号	PhoneNumber
email	E メール	EmailAddress

オブジェクトの表示や編集画面は，フィールド種別に連動して自動的に生成されるため，ユーザーがいちいち編集画面を作り込む必要はない．また，デフォルトの画面が気に入らない場合は，プラグインコードを作成することで，作り込むことも可能である．

6.1.2 オブジェクトのパス

オブジェクトストレージは一般的なファイルシステムのようにディレクトリ構造を持っている．オブジェクトのパス名は Unix のファイルシステムと同様に「/」（スラッシュ）区切りで表記される．Web ブラウザから Kompira オブジェクトへアクセスするには，単純にオブジェクトパスを URL として指定すればよい．たとえば，/servers/db_server1 というパスを持つオブジェクトにブラウザからアクセスするには，http://<Kompira サーバ>/servers/db_server1 という URL にアクセスすればよい．ここで，<Kompira サーバ>は対象のオブジェクトが格納されている Kompira サーバのホスト名もしくは IP アドレスを表すものとする．

6.1.3 データベーススキーマ

Kompira のオブジェクトストレージは実際には Kompira サーバで動作しているデータベース上に構築されている．ここで，オブジェクトの型を単純にデータベースの一つのテーブルとして実装してしまうと，ユーザーが新しい型を定義するたびに，データベースのスキーマを変更することになってしまい，現実的ではない．そこで，汎用のデータベーススキーマを用いることで，スキーマを変更せずに，新しい型定義を行えるようにしている．図 3 に Kompira オブジェクトストレージのデータモデルを示す．実際には，これ以外にもユーザーのパーミッション情報を格納するた

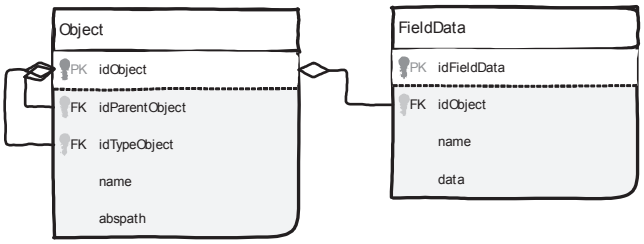

図 3　オブジェクトストレージのデータモデル

めのテーブルや，タイムスタンプ情報を格納するためのフィールドなどが含まれるが，本質的にはここに示したように非常に単純なスキーマにもとづいている．これは，Force.com のマルチテナント型アーキテクチャ[15]のためのスキーマを参考にして，Kompira に合わせたものである．

型オブジェクトやディレクトリ情報を格納するためのテーブルもデータモデルに含まれないことに注意してほしい．実際，型オブジェクトを表す型オブジェクトは以下のように通常の型オブジェクトとして定義されるデータに過ぎない．

フィールド名	フィールド表示名	フィールド種別
displayName	表示名	String
description	説明	Text
extend	拡張モジュール	String
fieldNames	フィールド名列	Array
fieldDispNames	フィールド表示名列	Array
fieldTypes	フィールド種別列	Array

6.1.4 仮想オブジェクト

ジョブフロー定義やサーバ情報など，Kompira で扱うオブジェクトのほとんどは，前述の汎用的なテーブルにデータとして格納されるが，Kompira にログインするためのユーザーアカウント情報やジョブフローの実行プロセス情報などの一部のデータについては，パフォーマンスや実装上の都合から，固有のテーブルやメモリ空間上に格納している．ただし，これらのデータについてもジョブフローや Web ブラウザからは，通常のオブジェクトと同様にアクセスしたい．そこで，Kompira では，仮想オブジェクトという Unix の proc ファイルシステムから着想を得た概念を用いて，ジョブフローから透過的にアクセスすることを可能にしている．

6.2 ジョブフロー言語

Kompira では，我々がジョブフロー言語と呼んでいる独自の外部 DSL を用いて，自動化の処理を記述する．ここでは，簡単な例を示しながら，ジョブフロー言語の機能について説明する．

6.2.1 リモートコマンドの実行

ジョブフロー言語では，一つひとつの処理の単位はジョブと呼ばれ，各ジョブを矢印記号（->，=>など）で結ぶことで，ジョブの実行順序を記述する．

リモートコマンドの実行にはコマンドジョブを使用し，実行したいコマンド文字列を角括弧（[ ]）で囲んで記述する．リモートのサーバ上（sample-server）で echo コマンドを実行し，結果を出力する簡単なジョブフローを示す．

```
{ __host__ = 'sample-server',
  __user__ = 'kompira', __password__ = 'kompira' |
  ['echo hello world'] -> print($RESULT) }
```

ここで，{ __host__ = …の部分はジョブの実行コンテクストを指定している部分で，実行ホスト名と実行ユーザー，パスワードを指定している．print()は組み込みのジョブであり，引数で与えられた値を Kompira のコンソールに出力する．$RESULT は特殊な変数で，直前のジョブの実行結果が格納される．ちなみに，実行コンテクストを指定しない場合，コマンドはジョブマネージャが動作しているサーバ上で実行される．

6.2.2 オブジェクトへのアクセス

Kompira オブジェクトは絶対パス，または相対パスを用いてジョブフローから参照することが可能である．また，オブジェクトのフィールド参照にはドット表記を用いる．以下は，サーバ一覧で定義されているサーバ情報を順次参照し，各サーバ上で hostname コマンドを実行し，結果をサーバ情報のフィールドに格納するジョブフローの例である．

```
{ for server in /sample/サーバ一覧 |
  [ __host__ = server.ipaddress, __user__ = server.user,
    __password__ = server.password ]
  -> ['hostname'] -> [$RESULT >> server.hostname] }
```

上の例の中で，for ブロックはサーバ一覧ディレクトリに含まれるオブジェクトを順次取り出して，server 変数に格納し，ブロック内（{と}で囲まれた部分）のジョブフローを繰り返し実行する．[$RESULT >> server.hostname]はデータ更新ジョブで，この場合，server オブジェクトの hostname フィールドに対して$RESULT に格納されている値を書き込む処理を行う．Kompira オブジェクトのデータに対しては，このように簡単な記法によって，ジョブフローから読み書きを行うことができる．

6.2.3 ジョブフロー／スクリプトジョブの呼び出し

あるジョブフローから別のジョブフローを呼び出すには，角括弧の中にジョブフローオブジェクトを指定すれば良い．ジョブフローにはパラメータを渡すことも可能である．以下に例を示す．

```
[/share/Library/サービス再起動:service='httpd']
 -> print('Apache を再起動しました')
```

ここで，service='httpd'の部分は，ジョブフロー（ここでは/share/Library/サービス再起動）に渡すパラメータを指定している部分である．

場合によっては，bash や perl などの既存のスクリプト言語を用いてジョブを作成し，それをジョブフローから呼び出したいこともある．既存のスクリプト言語を用いた処理を Kompira オブジェクトのスクリプトジョブとして作成しておくことで，こういったことが可能となる．呼び出しの形式はジョブフロー呼び出しと同様である．

Kompira エンジンはスクリプトジョブの実行前に，まずスクリプトデータを実行対象となるサーバへ転送する．ジョブマネージャ側では，そのデータを一時ファイルに書き出し，実行許可ビットを立ててから実行する．スクリプトの実行が完了するとジョブマネージャは，実行結果は Kompira サーバに返すとともに，作成した一時ファイルを削除する．

6.2.4 チャネルによるイベントの取得

障害発生など外部からのアラートを契機にジョブを実行する例を示す．

```
</system/channels/Alert>
-> { case $RESULT.message |
    'PROBLEM': [./障害復旧手順: $RESULT.data]
    'OK': print('障害が回復しました') }
-> self()                # イベントを処理したら最初に戻る
```

/system/channels/Alert は Kompira のチャネルオブジェクトであり，外部の監視サーバなどからのアラートを受け付ける特別な Kompira オブジェクトである．監視サーバ側では，障害発生時に，Kompira サーバのメッセージキューに対して，障害発生のイベント情報を JSON 形式で送信する．すると，Kompira のジョブフロー側ではチャネルからそのイベントを受け取ることができる．上記の例の中で，山括弧（<と>）はイベントジョブを示しており，何かイベントを受け取るまで，ジョブフローの実行を待機する．受信したイベントのデータは$RESULT 変数に格納される．

6.3 ジョブフローのコンパイルと実行エンジン

ジョブフローは一旦，中間言語に変換されてから，Kompira の実行エンジン（スタック方式のインタプリタ）によって解釈実行される．中間言語へのコンパイルは，ユーザーがジョブフローを編集し終わり，保存した時に自動的に行われる．このとき，構文エラーなどが見つかった場合には，エラーを報告し，そのジョブフローは実行できないようになる．実行エンジンは複数のジョブフロープロセスの同時実行を許すため，プロセス管理の機能も実装している．たとえば，あるプロセスがリモートコマンドの実行待ち状態になると，別のプロセスに実行を切り替えるといった処理を行う．

7. 適用事例

この章では，Kompira の適用事例をいくつか紹介する．

7.1 バッチ処理の一元管理

各サーバで行う定常的な作業は，bash や perl のようなスクリプトによって自動化はされているが，サーバ台数が多く，また，それぞれのスクリプトも，各サーバ上に散らばっていて，どのサーバでどのスクリプトが使われ，動作しているのか管理されていない状態であった．そこで，これらのスクリプトを Kompira のスクリプトジョブとして定義することで，Kompira から一括してスクリプトを実行することが可能となった．また，スクリプトも Kompira サーバで一元管理されるので，システム全体で，どのような運用が行われているのか把握しやすくなった．

7.2 不要なファイルの削除

あるシステムでは，ユーザーがサービスを利用するたびに，ある種のファイルが生成されるようになっている．時間の経過とともに，生成されるファイルがディスク領域を圧迫してくる．オペレータは監視ツールからディスク領域が残り少ないとの通知を受けるたびに，対象サーバにログインして，ファイルシステムを調べて，古くなった不要なファイルを削除するといった作業に追われていた．そこで，Kompira 側で監視ツールからアラートを受けるようにしておき，そのアラートを契機として，対象マシンにログインして，削除対象となるファイルの一覧を洗い出し，結果を担当者にメールする部分を Kompira で自動化している．将来的には不要なファイルの削除までも含めて Kompira のジョブフローで自動実行させる予定である．

7.3 プロセス一覧表の自動作成

弊社では，顧客から運用監視を請け負っている各サーバで，実際にどのようなプロセス（サービス）が動作しているかといった情報を管理するために，Excel を用いて，各サーバのプロセス情報の一覧を作成していた．大量のサーバを管理している場合，各サーバにいちいちログインして ps コマンドで確認していくのは，労力の要する作業となる．そこで，Kompira でサーバ一覧に定義された各サーバにログインしてプロセス情報を収集し，Excel で読み込めるような CSV 形式のファイルを出力するようなジョブフローを定義することで，これらの作業を自動化した．

7.4 公式サイトのコンテンツ更新

Kompira 公式サイトのコンテンツは外部のデザイナに作成を依頼している．デザイナはサイトの内容を更新すると，コンテンツファイル一式を特定のディレクトリにアップロードする．デザイナからアップロードの連絡を受けると，これらの最新ファイルを一度，ステージング環境にコピーする．そして，内容確認のタスクを作成，承認を得られると，既存サイトの内容をバックアップし，公式サイトを最新のものに更新するという一連の作業を自動化している．

8. 検討と考察

ここでは，Kompira を導入することによって得られる効果と限界について検討する．

8.1 自動化に適しているものとそうでないもの

Kompira で運用の自動化を行うには，まず，運用手順がしっかりと定型化されていることが前提であり，それをジョブフローに落とし込む必要がある．このため，手順があいまいなものや，あまりに複雑な手順なものは，ジョブフロー自体の作成とテストに手間がかかってしまい，自動化によって得られる効果を上回ることが考えられる．単純な手順で実行頻度が高いものほど Kompira での自動化に適している. 弊社監視センターでは, 毎月の作業工数を集計し，内訳の分析を行っている．それによると，障害時のアプリケーション再起動や，エスカレーションなど手順書通りの作業である 1 次障害対応と，サーバ証明書の更新や，DNS の設定変更，ユーザー追加など，定型的な運用作業にかかる時間が全体の約 65%を占めている．この部分はほとんどが Kompira で自動化できると期待できる．さらに，SSL サーバ証明書の更新や，DNS の設定変更など，一般的な現場でよくみられる運用手順は，あらかじめ作成済みのジョブフローをライブラリとして提供することで，自動化の導入にかかる手間を引き下げることも可能だろう．

8.2 既存ツールと連携することの利点と欠点

Kompira は既存の運用ツールと連携して使うことを想定し，連携のための手順を自動化することにフォーカスすることで，Kompira 自体は小さくシンプルなツールとなっている．これは一つの利点であるが，本格的な運用に活用するためには，実際には様々なツールと連携してシステムを構築する必要があり，連携対象の周辺ツールを含めた全体としては複雑で大きなものになってしまう可能性がある．連携する対象のツールが増えたときの問題点としては，ユーザーインタフェースが，それぞれのツールごとにバラバラに提供されるため，統一感が無くなるということが挙げられる．また，ログイン画面だけでも，ツールの数だけあるため，それぞれのツール上でユーザーアカウントの設定を行い，いちいちログインしなくてはならないという問題もある．ただし，これについては，OAuth などの利用である程度は解決できる可能性はある．

もう一つの問題としては，ある種の機能の重複がある．たとえば，資産管理ツールと，サーバ監視ツールを一緒に使うことを考えた場合，たいていの場合どちらのツールもサーバ情報のためのデータベースを備えており，それぞれにデータを登録しておく必要がある．Kompira で，これらのデータ間の連携を自動化するジョブフローを作成することは可能だが，同じリソースに関するデータが複数のデータベースにまたがって存在するのは，整合性の問題を引き起こしやすい．

9. まとめ

本稿では，IT システムの運用現場が抱える問題を指摘し，その上で，これらの問題を解決するための 1 つのツールとして，我々が開発している運用自動化のためのプラットフォームである Kompira について紹介した．

Kompira は定型的な手順として記述できるあらゆる運用保守業務を自動化することを究極の目標として開発を進めているが，現バージョンの Kompira では，対象となる機器は SSH でリモートログインしてコマンド実行できるものに限られている．実際の運用手順の中には，コマンドを実行するだけではなく，ウェブブラウザを起動して，サイトが正しく動作しているかログインして確認するといったものや，処理のためにアプリケーションを起動し，それに対して操作を行うといったものもある．これらの処理を Kompira のジョブフローでどのように記述し，どのように実行していくのが良いかは, 今後の検討課題の 1 つである．

最後に，我々のツールが運用現場を担っている多くのエンジニアに使われ，運用現場をより創造的で，歓びに溢れる場へ変えていく手伝いができれば望外の幸せである．

謝辞 我々の主催している運用自動化研究会に参加して下さった方々には，Kompira に関する数多くの貴重な意見や要望を賜りました．また，フォースクーナ株式会社の同僚諸氏は，Kompira の構想段階から今日にいたるまで，温かい期待と応援を寄せていただき，筆者らが開発を進めていく上での大きな励みになりました．この場をお借りして感謝いたします．

参考文献

1) Kompira 公式サイト, http://www.kompira.jp/
2) 波多野，見えない「運用」ー疲弊する現場,
http://thinkit.co.jp/story/2010/12/02/1903
3) ファーストサーバ，調査報告書,
http://support.fsv.jp/urgent/pdf/fs-report.pdf
4) DevOps, http://www.devopsdays.org/
5) 統合システム運用管理 JP1 ,
http://www.hitachi.co.jp/Prod/comp/soft1/jp1/
6) Hinemos, http://www.hinemos.info/
7) Crane, https://www.oss.ecl.ntt.co.jp/ossc/oss/r_crane.html
8) IT Leaders, http://it.impressbm.co.jp/e/2011/01/24/3281
9) Puppet, http://puppetlabs.com/
10) Chef, http://www.opscode.com/chef/
11) Capistrano, https://github.com/capistrano/capistrano/wiki
12) Fabric, http://docs.fabfile.org/
13) RightScale, http://www.rightscale.com/
14) Jenkins, http://jenkins-ci.org/
15) Force.com のマルチテナント型アーキテクチャ,
http://www.developerforce.com/media/ForcedotcomBookLibrary/Force.com_Multitenancy_WP_101508_JP.pdf